OIOIDWS
Overview and Installation

Content

1 Intruduction and overview of OIODIWS

This package contains a sample implementation of the specs for OIOIDWS (identity-based webservice). It encapsulates subelements from SAML, WS-Trust and Libery Basic SOAP Binding.

An identity-based webservice, is a service that operates on behalf of a given user. It might be a service that grants a doctor access to medical information, or another that allows banks to exchange customer information.

OIO identity-based webservices (OIOIDWS) is a series of webservice profiles, developed and maintained by IT & Telestyrelsen, which outlines a standardized method for webservice-consumers to access webservice-providers on behalf of the end-user, using that users identity in a secure way.

OIOIDWS enables services to make personal data available, in a secure and standardized way, which is a prerequisite for automation of processes across different institutions, organizations and domains.

The reference implementation covers a specific scenario with an identity-based webservice. Here a user, using a web browser, logs on to a web-application, where the web-application needs to access a webservice at another organization, using the user's identity.

1.1 Purpose of the OIOIDWS.JAVA package

The purpose of the java package is to offer the set of components required to run a small example of a OIODIWS-powered SOAP-call on the Java-platform, based on SUN's reference implementation of the required components and libraries.

1.2 Purpose of the OIOIDWS.NET package

The purpose of the .NET package is to demonstrate how Microsofts tools are used, and configured to perform a OIOIDWS call.

1.3 Prerequisites

The intended audience of this document is people experienced with webservices and system-to-system integration in general.

It is recommended to read both the Java and .NET sections, to ensure a common understanding of how OIOIDWS issues are addressed on both platforms.

1.4 The contents of the Java package

The OIOIDWS.JAVA package contains 3 modules, which are described in the installations section of this document.

- **Web Service Consumer (WSC)**
  - A SAML 2.0 Service Provider, which accesses the STS and the WSP web service.

- **Web Service Provider (WSP)**
  - A simple "echo service", requiring a valid SAML token from the STS.

- **Security Token Service (STS)**
  - An OIOSAML Demo STS, following the WS-Trust 1.3 standard, able to issue service-specific tokens used in the sample.

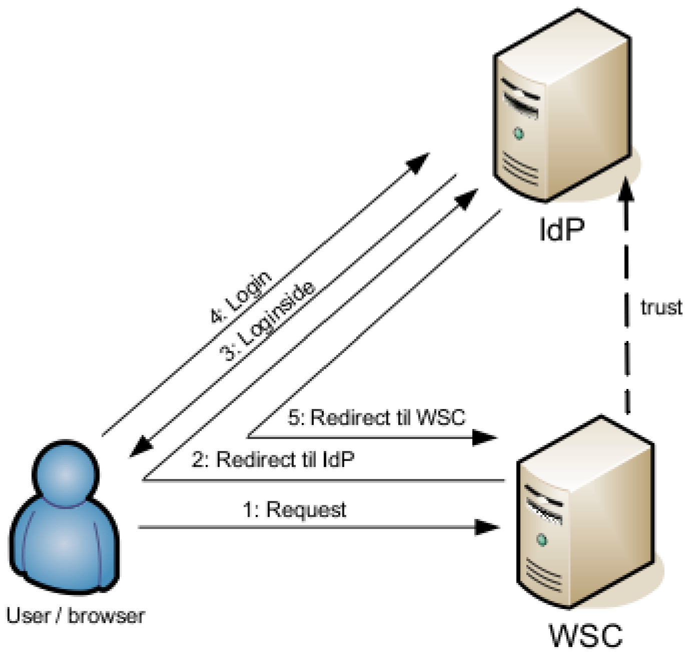
These modules demonstrate the concepts of OIODIWS on the Java platform. It is important to add that, that the STS require a SAML assertion issued by a trusted IdP. Though the IdP is not part of OIOIDWS as such, for compleness, the OIOIDWS.java package contains instructions on configuring an IdP.

1.4.1 Overview and scenarios

This section outlines and explains the interactions between the modules, and how trust are established between them.

Bootstrap token

The flowchart below is based on certain elements in the "fællesoffentlige brugerstyring", where the scenario is bootstrapped in a webapplication associated with an IdP. Here the Webservice Client (WSC) is both a WSC in the OIOIDWS context, and a Service Provider in the OIOSAML context.



The user logs onto the WSC using the associated IdP.

1: The user accesses the WSC
2,3,4: The user is redirected to the IdP and performs a logon

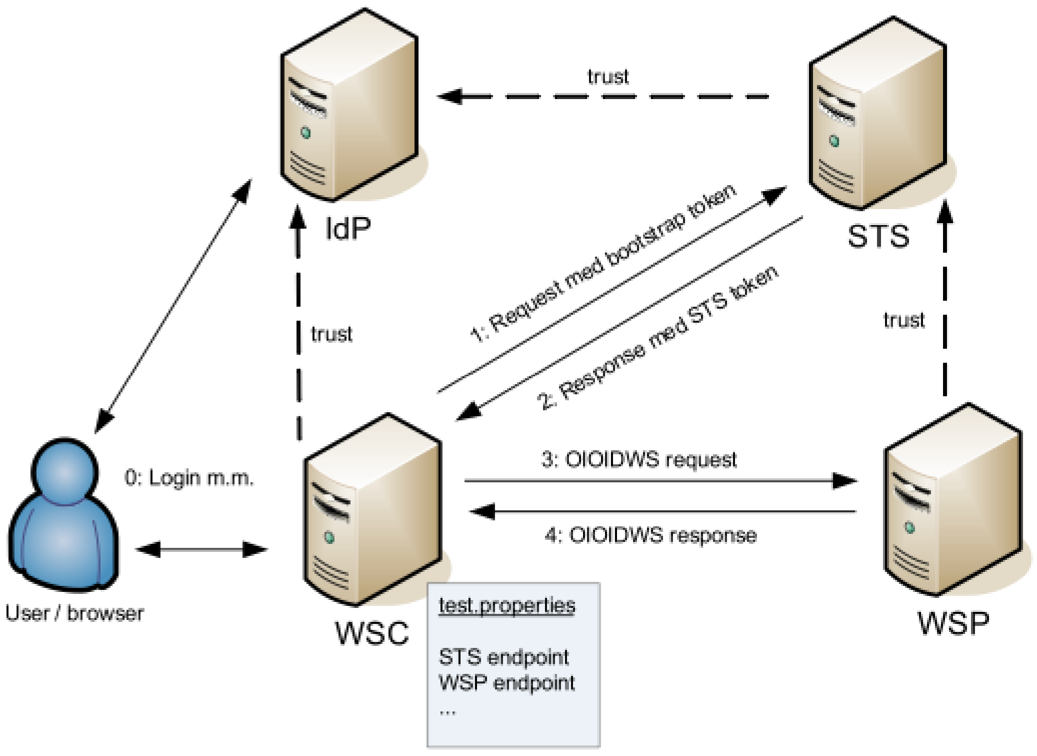
5: The user is redirected back to the WSC

The role of the IdP is to handle logins (single-signon if more than one application is associated with the same IdP) and to issue signed SAML-tokens, used in the final call (5) in the flow above.

The SAML token ensures that WSC that the login was performed correctly, and that the user is who he/she claims he/she is. In addition, the token contains a bootstrap-token, which is used in later authentication when calling external webservices (the OIOIDWS part).

The OIOIDWS call

To perform the webservice-cal between the WSC and the webservice provider (WSP), the bootstrap token mentioned earlier, needs to be exchanged for a service-specific token. For this purpose, the STS must be used. The complete scenario ends up as below:



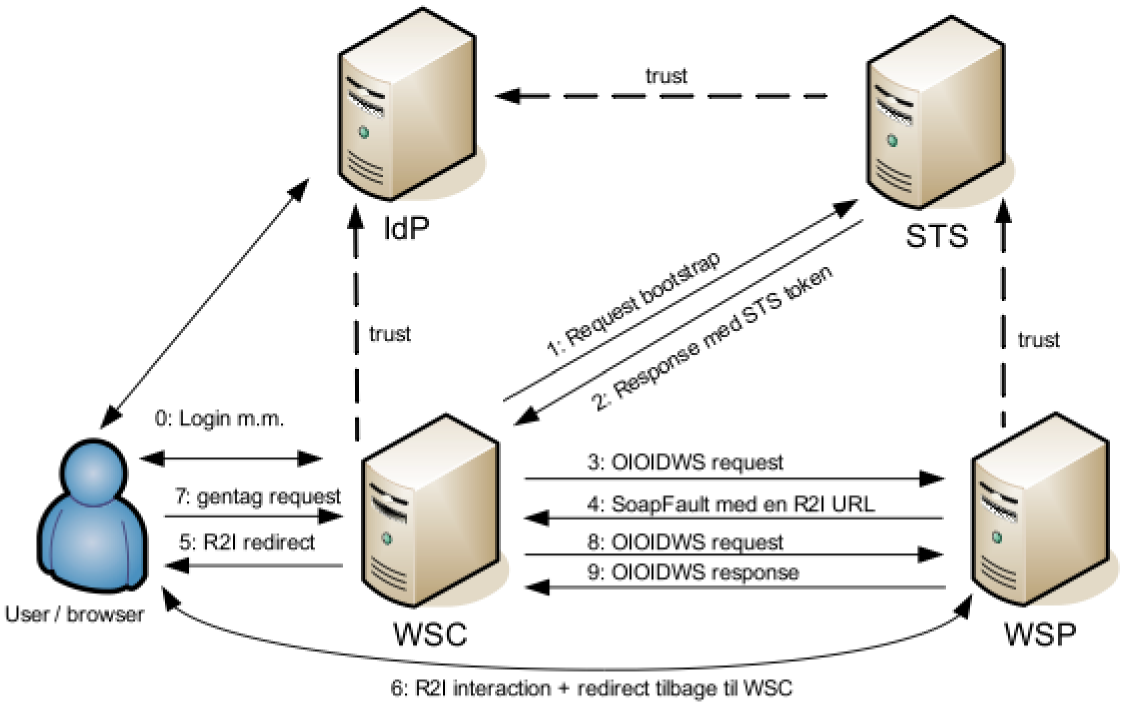
0: The user performs the login mentioned in the earlier flowchart
1,2: WSC exchanges the bootstrap-token for a service-specific token at the STS
3,4: The WSC uses the service-specific token to access the WSP

The servicespecific token is a SAML token that contains, besides the identifier of the WSP, a timelimit and a serialnumber, ensuring that the service can be accessed only once, and only in a given timeperiod.

In the Java package, SimpleSAMLPhP is used as the IdP, and in the .NET package, it is an ADFS that is used.

Request to interact

The example outlined above is the simplest way to perform the operation. Another variant is called “Request to Interact (R2I)”, which involves the end-user in an additional check. The flow in R2I looks like this:



The above scenario is also implemented in the reference package.

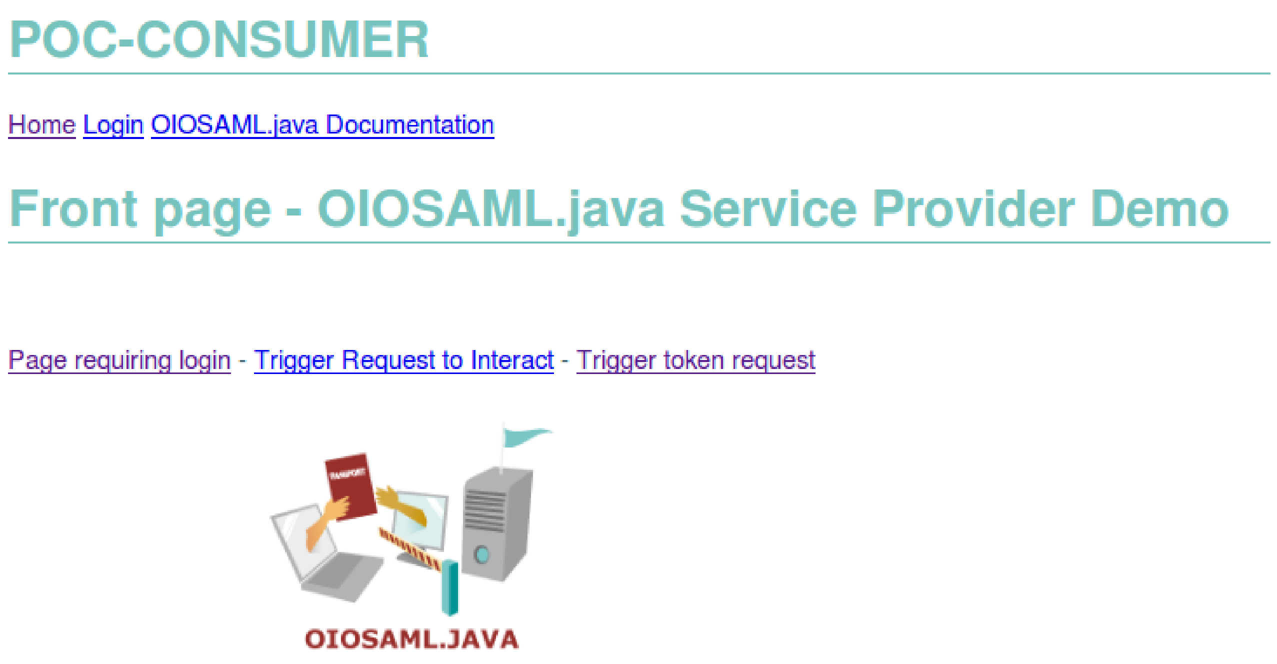
Trust

Trust between the following parties is required for the above to work.

- WSC and IdP: The WSC needs to trust the SAML tokens issued by the IdP.
- WSP and STS: The WSP needs to trust the tokens issued by the STS.
- STS and IdP: The STS needs to trust the bootstrap token issued by the IdP.

Scenarios

The WSC in the reference implementation is, as mentioned, a web application. When accessing the WSC in a browser, the following screen is shown:



POC-CONSUMER

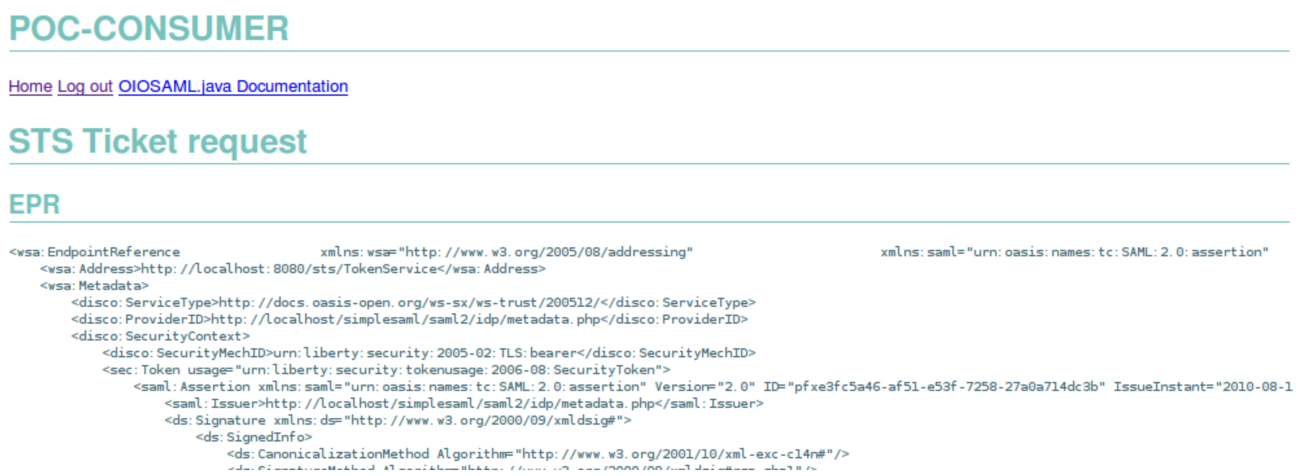
Home Login OIOSAML.java Documentation

Front page - OIOSAML.java Service Provider Demo

Page requiring login - Trigger Request to Interact - Trigger token request

The link ”Page requiring login” will redirect to the IdP, and after a successull login, the IdP-issued SAML token is presented in the browser.

Below the presentation of the SAML token, the link ”Trigger token request” is shown, which will exchange the token for a service-specific token at the STS:



POC-CONSUMER

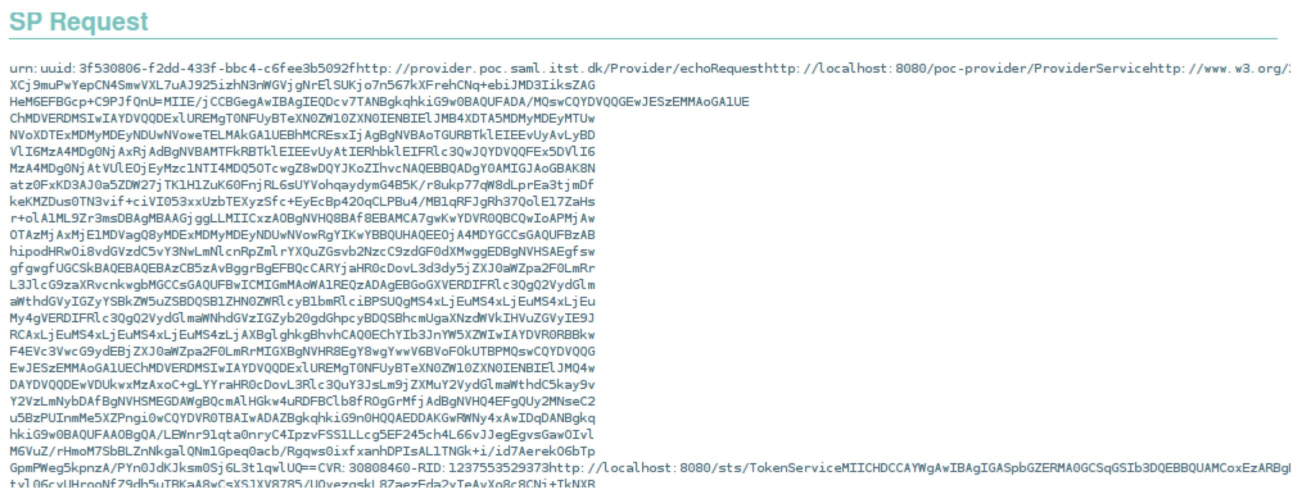
Home Log out OIOSAML.java Documentation

STS Ticket request

EPR

```
<wsa:EndpointReference                  xmlns:wsa="http://www.w3.org/2005/08/addressing"                      xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    <wsa:Address>http://localhost:8080/sts/TokenService</wsa:Address>
    <wsa:Metadata>
        <disco:ServiceType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/</disco:ServiceType>
        <disco:ProviderID>http://localhost/simplesaml/saml2/idp/metadata.php</disco:ProviderID>
        <disco:SecurityContext>
            <disco:SecurityMechID>urn:liberty:security:2005-02:TLS:bearer</disco:SecurityMechID>
            <sec:Token usage="urn:liberty:security:tokenusage:2006-08:SecurityToken">
                <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" Version="2.0" ID="pfxe3fc5a46-af51-e53f-7258-27a0a714dc3b" IssueInstant="2010-08-1
                    <saml:Issuer>http://localhost/simplesaml/saml2/idp/metadata.php</saml:Issuer>
                    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                        <ds:SignedInfo>
                            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
```

Below this, the link ”Perform token WS request” is shown, which will perform the web-service request (the OIOIDWS part), and the result will be shown in the browser:



SP Request

```
urn:uuid:3f530806-f2dd-433f-bbc4-c6fee3b5092fhttp://provider.poc.saml.itst.dk/Provider/echoRequesthttp://localhost:8080/poc-provider/ProviderServicehttp://www.w3.org/
XCj9muPwYepCN4SmwVXL7uAJ925izhN3nWGVjgNrElSUKjo7n567kXFrehCNq+ebiJMD3IiksZAG
HeM6EFBGcp+C9PJfQnU=MIIE/jCCBGegAwIBAgIEQDcv7TANBgkqhkiG9w0BAQUFADA/MQswCQYDVQQGEwJESzEMMAoGA1UE
ChMDVERDMSIwIAYDVQQDExlUREMgT0NFUyBTeXN0ZW10ZXN0IENBIElJMB4XDTA5MDMyMDEyMTUw
NVoXDTExMDMyMDEyNDUwNVoweTELMAkGA1UEBhMCREsxIjAgBgNVBAoTGURBTklEIEEvUyAvLyBD
VlI6MzA4MDg0NjAxRjAdBgNVBAMTFkRBTklEIEEvUyAtIERhbklEIFRlc3QwJQYDVQQFEx5DVlI6
MzA4MDg0NjAtVUlEOjEyMzc1NTI4MDQ5OTcwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAK8N
atz0FxKD3AJ0a5ZDW27jTK1H1ZuK60FnjRL6sUYVohqaydymG4B5K/r8ukp77qW8dLprEa3tjmDf
keKMZDus0TN3vif+ciVI053xxUzbTEXyzSfc+EyEcBp420qCLPBu4/MB1qRFJgRh37QolE17ZaHs
r+olA1ML9Zr3msDBAgMBAAGjggLLMIICxzAOBgNVHQ8BAf8EBAMCA7gwKwYDVR0QBCQwIoAPMjAw
OTAzMjAxMjE1MDVagQ8yMDExMDMyMDEyNDUwNVowRgYIKwYBBQUHAQEEOjA4MDYGCCsGAQUFBzAB
hipodHRwOi8vdGVzdC5vY3NwLmNlcnRpZmlrYXQuZGsvb2NzcC9zdGF0dXMwggEDBgNVHSAEgfsw
gfgwgfUGCSkBAQEBAQEBAzCB5zAvBggrBgEFBQcCARYjaHR0cDovL3d3dy5jZXJ0aWZpa2F0LmRr
L3JlcG9zaXRvcnkwgbMGCCsGAQUFBwICMIGmMAoWA1REQzADAgEBGoGXVERDIFRlc3QgQ2VydGlm
aWthdGVyIGZyYSBkZW5uZSBDQSB1ZHN0ZWRlcyB1bmRlciBPSUQgMS4xLjEuMS4xLjEuMS4xLjEu
My4gVERDIFRlc3QgQ2VydGlmaWNhdGVzIGZyb20gdGhpcyBDQSBhcmUgaXNzdWVkIHVuZGVyIE9J
RCAxLjEuMS4xLjEuMS4xLjEuMS4zLjAXBglghkgBhvhCAQ0EChYIb3JnYW5XZWIwIAYDVR0RBBkw
F4EVc3VwcG9ydEBjZXJ0aWZpa2F0LmRrMIGXBgNVHR8EgY8wgYwwV6BVoFOkUTBPMQswCQYDVQQG
EwJESzEMMAoGA1UEChMDVERDMSIwIAYDVQQDExlUREMgT0NFUyBTeXN0ZW10ZXN0IENBIElJMQ4w
DAYDVQQDEwVDUkwxMzAxoC+gLYYraHR0cDovL3Rlc3QuY3JsLm9jZXMuY2VydGlmaWthdC5kay9v
Y2VzLmNybDAfBgNVHSMEGDAWgBQcmAlHGkw4uRDFBClb8fROgGrMfjAdBgNVHQ4EFgQUy2MNseC2
u5BzPUInmMe5XZPngi0wCQYDVR0TBAIwADAZBgkqhkiG9n0HQQAEDDAKGwRWNy4xAwIDqDANBgkq
hkiG9w0BAQUFAAOBgQA/LEWnr91qta0nryC4IpzvFSS1LLcg5EF245ch4L66vJJegEgvsGaw0Ivl
M6VuZ/rHmoM7SbBLZnNkgalQNm1Gpeq0acb/Rgqws0ixfxanhDPIsAL1TNGk+i/id7Aerek06bTp
GpmPWeg5kpnzA/PYn0JdKJksm0Sj6L3t1qwlUQ==CVR:30808460-RID:1237553529373http://localhost:8080/sts/TokenServiceMIICHDCCAYWgAwIBAgIGASpbGZERMA0GCSqGSIb3DQEBBQUAMCoxEzARBg
tyl06cyUHrooNf79db5uTBKaA8wCsXSJXV8785/UQyezgskL87aezFda2yTeAyXo8c8CNj+TkNXR
```

The linke ”Trigger request to interact” will perform a R2I-call from the WSC to the WSP. The WSP will redirect the browser to a page, where the user can enter a ”secret”. When this is completed, the webservice request is finished, and the entered secret is carried back to the WSC.

1.5 The content of the .NET package

1.5.1 Sub-components

The demo application is a small WPF/WCF-based client, that fetches tokens for use with the echo-webservice. The project is split into a series of Visual Studio 2008 projects:

Bindings
Uses WIF (Windows Identity Foundation) to implement a series of helper-classes, used for creating WCF bindings to access the STS.

Client
Unittests that uses a self-issued bootstrap token to test calls to both a .NET and a Java based webservice, through a java STS. Both SSL and non-SSL calls are made.

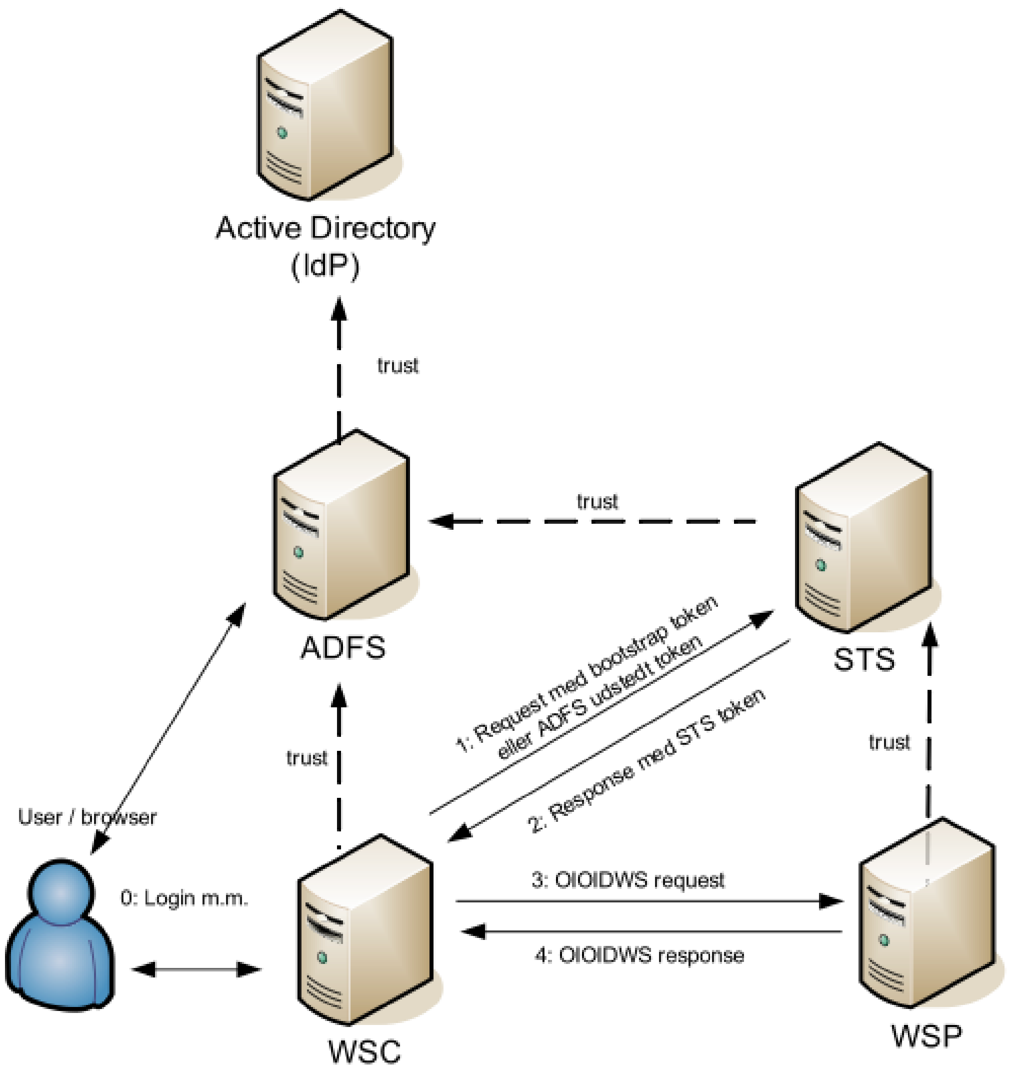
EchoService
WCF based interface for the echo webservice. Used by both the service-client and the implementation of the serviceprovider.

EchoWebServiceProvider (WSP)
WCF implementation of the Echo service. Authorization of calls to the provider are done by SericeProviderBinding from the Bindings assembly.

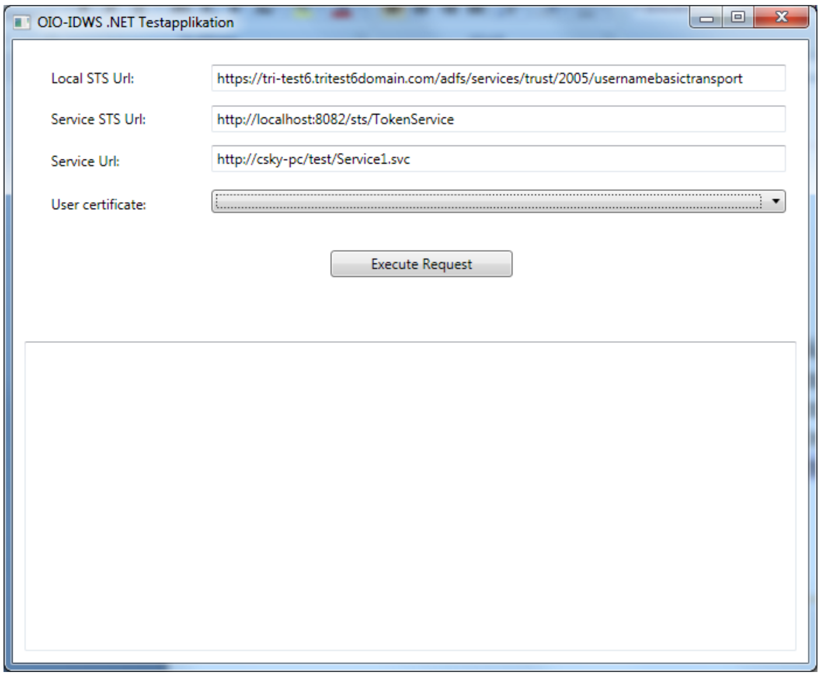
WPFClient (WSC)
WPF based .NET GUI demo application that uses WCF and WIF to issue tokens, and uses these, through the Bindings component, to perform webservice calls.

No .NET STS is included in the package, but the OIOSAML Trust Demo STS from the OIOSAML.Java package can be used.

The scenario for the .NET package is identical to the one mentioned in section 2.1.2 in this document. The WPFClient application contains codesamples for issuing bootstrap tokens, and or issuing a token from an ADFS 2.0 server. The ADFS server performs autorization by using an AD server, which acts as an IdP in this setup. The bootstrap-token, or the ADFS-issued token, is exchanged at the STS for a service specific token, which can be used to access the Echo webservice.



1.5.2 **The rich .NET client**



Local STS Url: https://tri-test6.tritest6domain.com/adfs/services/trust/2005/usernamebasictransport

Service STS Url: http://localhost:8082/sts/TokenService

Service Url: http://csky-pc/test/Service1.svc

User certificate:

This demo application is an example on how a rich .Net application can use OIOIDWS. A series of parameters are available for calling the Echo service:

- Local STS Url: Url for the IdP used to issue tokens. If left blank, a self-issued token will be used.
- Service STS Url: Url for the STS, which can exchange tokens to service specific tokens.
- Service Url: Url for the Echo webservice
- User certificate: The user certificate (installed in CAPI) used for authentication against the IdP.
- By clicking ”Execute Request”, a token is retrieved from the IdP, exchanged at the STS and used for the call against the Echo service. The result of each call is shown in the window.

1.6 Andre scenarier

The above mentioned scenarios are just examples on how OIOIDWS can be used. Another common usage, is a rich application that is running on a machine already authenticated against a local AD. In this situation, the STS might use a Kerberos-token as the bootstrap-token. Hence the term bootstrap token is used as a general term for something exchanged at the STS. Likewise a more complex setup could have several STSes for several WSPs. On STS might exchange a token granted by one STS to use for another service.

Common for all the scenarios, is the principal that a service calls another service, on behalf of the end-user, and that credentials are carried by a STS-issued “ticket”.

More scenarios can be found at the following location:http://digitaliser.dk/resource/416476/artefact/OIO+IDWS+Scenarios+1.0.pdf

2 Installation Guide

This section describes how to install and configure the OIOIDWS components and run a few manual tests based on them.

The test setup consists of the following components:

1. Secure Token Service (STS), running in Java using custom code
2. WebService Provider (WSP), running either Java (custom code) or .NET (WIF)
3. WebService consumer (WSC), in this case implemented as a Java-based web application or a rich .Net client (WPF/WIF/WCF)
4. An Identity Provider (IdP), in this case the freely available SimpleSAMLphp for the java platform and an ADFS for the .Net platform

The manual tests show that

- the WSC is protected by an IdP login
- the WSC can retrieve tokens from the STS
- the WSC client can invoke the WSP service
- the WSC and WSP support Request to Interact (R2I)

There is also a suite of automated integration tests that provide more detailed - though not complete - testing of a number of details in this setup. However, the detailed setup for running those tests is not described here.

2.1 Configuring the Java components

The following must be installed: JDK 6, Subversion (this guide assumes the command line version, substitute as appropriate if using something like TortoiseSVN).

Install an Application Server

The application server used is GlassFish v2. Download from https://glassfish.dev.java.net/ and install and start it.
This package is tested on Glassfish 2.2.1.

Install the STS

The STS consists of a regular WAR file which can be installed into GlassFish/Tomcat or any other servlet container. Download the most recent STS from http://digitaliser.dk/group/42063

Limitations:

- Only supports WS-Trust 1.3
- Supports both SOAP 1.1 and 1.2
- Only supports SAML 2.0 TokenType
- All SAML Assertion attributes are copied from the request assertion
- Requests must contain a SAML Assertion in OnBehalfOf
- Requests must be signed

- There is no real error handling
- The AppliesTo value will be copied without further checking

These limitations means that the STS is in no possible way production ready, but this not important for this project where the goal is to show how to establish WSC and WSP functionality. Thus, the STS is just needed for testing of the scenarios.

Before deploying the STS, it must be configured with an EntityID and a certificate. Do this by creating a new file in `~/.oiosaml (C:\Documents and Settings\<username>\.oiosaml` on Windows XP or `C:\Users\<username>\.oiosaml` on Windows 7) called `sts.properties` containing the following:

```
sts.entityId=http://sts.oiosaml.net
sts.certificate.location=TestVOCES1.pkcs12
sts.certificate.password=Test1234
```

If this file is missing, the deployment will fail with an error saying "IllegalStateException: System not configured".

Modify the values according to your local settings. The certificate location is relative to the `sts.properties` file itself and can be either a JKS keystore or a PKCS12 file.
For test certificates, go to https://www.certifikat.dk/export/sites/dk.certifikat.oc/da/developer/eksempler/.

Unzip the archive and deploy the war file. For deployment to GlassFish, do as follows:

- Open http://localhost:4848 in a browser
- Login (default administrator login is admin/adminadmin)
- Click on Applications in the menu, then on Web Applications
- Click Deploy...
- Choose the sts.war file and click OK

Restart GlassFish. Check the GlassFish logs at `domains/domain1/logs/server.log` to check that everything is running - one of the last lines should contain "`Configured OIOSAML to .../.oiosaml/sts.properties`".

Install the WebService Provider

The WebService Provider is a standard JEE web application. It has been tested with GlassFish only and it requires Metro/WSIT version 2.0 (not 2.0.1 – there is a problem with signature validation with this version). Download the most recent version from http://digitaliser.dk/group/705156 and unzip the file. Deploy the poc-provider.war file to GlassFish by following the same instructions as for the STS. This can be done into either the same GlassFish instance, or with an instance on another host.

The only required configuration for the WSP is that a number of certificates and keys must be installed:

- The STS certificate must be present in the truststore
- The root CA certificate for the WSC certificates must be present in the truststore
- The private key and certificate must be present in the server's keystore

The STS certificate:

The STS certificate must be present in the truststore. First, list the contents of the STS keystore file in order to find the alias of the key.

```
keytool -list -keystore TestVOCES1.pkcs12 -storetype pkcs12
```

The output should look similar to this:

```
Keystore type: PKCS12
Keystore provider: SunJSSE

Your keystore contains 1 entry

2, Jul 5, 2010, PrivateKeyEntry,
Certificate fingerprint (MD5): 75:D0:2A:21:A0:68:BB:1E:6A:E9:F2:AE:33:16:31:16
```

The alias is first part of the second last line (in this case the "2" that starts the line "2, Jul 5, 2010, PrivateKeyEntry"). Note that this is a "PrivateKeyEntry". What we need is a certificate, which can be generated and imported with the following commands.

```
keytool -export -rfc -keystore TestVOCES1.pkcs12 -storetype pkcs12 -alias
"<alias read from the output of the previous command>" > sts.crt
```

(if necessary copy `sts.crt` to the host of the WebService Provider)

```
cd <glassfish_home>/domains/domain1/config
keytool -import -trustcacerts -keystore cacerts.jks -file sts.crt -alias sts
```

The first command requires the password for the keystore (e.g. "Test1234"). The last command requires the password for the truststore of the Glassfish domain, which is "changeit" by default. The command also prompts for confirmation that the certificate should be trusted - make sure to change the default answer from "no" to "yes".

The root certificate

The root CA certificate for the WSC certificates must be present in the truststore. For OCES test certificates, the CA certificate can be found
at https://www.certifikat.dk/export/sites/dk.certifikat.oc/da/developer/eksempler/.
(get the ca certificate and save it to `cacert.crt`)

```
cd <glassfish_home>/domains/domain1/config

keytool -import -trustcacerts -keystore cacerts.jks -file cacert.crt -alias
cacert
```

Again, the last command requires the password for the truststore of the Glassfish domain which is "changeit" by default. The command also prompts for confirmation that the certificate should be trusted - make sure to change the default answer from "no" to "yes".

The private key

The private key must be copied to the server's keystore and must have a "voces" alias.

First, get the alias of the key in the existing file:

```
keytool -list -keystore TestVOCES1.pkcs12 -storetype pkcs12
```

As in the section describing the STS certificate import, the alias is the first part on the 2nd last line.

Then, import the key into the server keystore:

```
cd <glassfish_home>/domains/domain1/config

keytool -importkeystore -srckeystore TestVOCES1.pkcs12 -destkeystore
keystore.jks -srcstoretype pkcs12 -destalias voces -srcalias "<alias>"
```

NOTE 1: The keytool command can behave a bit strange. Even though the alias found in the keystore listing above was for example "2" it may not accept this as srcalias. Given that the source keystore contains only the relevant key, simply try using "1" instead, if you experience this problem - it has been seen to work several times.

The import can be verified by listing the contents of the keystore.jks and verifying that it contains an entry with the alias "voces":

```
$ keytool -list -keystore keystore.jks

Enter keystore password:

Keystore type: JKS
Keystore provider: SUN

Your keystore contains 2 entries

s1as, Jun 8, 2010, PrivateKeyEntry,
Certificate fingerprint (MD5): D5:45:32:04:5A:1B:13:84:BA:76:85:AF:4C:06:AA:A7
voces, Jul 5, 2010, PrivateKeyEntry,
Certificate fingerprint (MD5): 75:D0:2A:21:A0:68:BB:1E:6A:E9:F2:AE:33:16:31:16
```

NOTE 2: Glassfish requires that the keys in the keystore all have the same password as the master password of the keystore itself - otherwise the server will fail during startup of the domain with a log entry saying "Cannot recover key" (which just means that it was not possible to find/use a key with the given alias and password). The password can be changed using the following command which will prompt for the keystore password, the (old) key password and the new key password. If the password is changed multiple times it can be a little confusing, because keytool does not prompt for the old key password if the keystore password is also valid as key password.

```
keytool -keypasswd -keystore keystore.jks -alias voces
```

It might be necessary to restart GlassFish after installing the keys and certificates.

```
<glassfish_home>/bin/asadmin stop-domain
<glassfish_home>/bin/asadmin start-domain
```

Test that the application is installed correctly by retrieving the WSDL file. To do this, open http://localhost:8080/poc-provider/ProviderService?wsdl in a browser (edit hostname and port number according to the local installation if the defaults have been changed).

Install the WebService Consumer

The WebService Provider is a standard JEE web application. Download the most recent version from http://digitaliser.dk/group/705156and unzip the file. Deploy the poc-consumer.war file to GlassFish or to another Servlet container by following the same instructions as for the STS. This can be done in the same GlassFish instance, or on an instance on another host.

Configure the application for an Identity Provider (IDP)

It is a prerequisite for this step that an IDP is installed and available - the IDP must support SAML 2.0 Assertions (urn:oasis:names:tc:SAML:2.0:assertion version 2.0), but it is out of scope for this document to describe how to install and configure one. This scenario has been tested with SimpleSAMLphp which is freely available from http://simplesamlphp.org.

The IdP should be configured to map from a given STS Entity ID to a TokenService URL of an STS, and to provide it as an EndPointReference in the User Assertion. This is the value which will be inserted into the Issuer field of the generated assertion, because the STS is a very simple implementation that just uses the IDP to issue a SAML token. In production mode the IDP will not have any associations to the STS - the STS must however know which IDP's it trusts.

The OIOSAML.java filter is configured for the application by performing the following steps:

- Download or acquire the metadata for the IdP and save it in a file (e.g. idp-metadata.xml)
- Enter the URL http://localhost:8080/poc-consumer/saml/configure, which - for an unconfigured application - will yield a page for providing the configuration. If the page claims that the application is already configured, but you wish to reconfigure, then remove (i.e. either delete or move) the directory `~/.oiosaml-poc-consumer`, restart the application and try again.
- Fill in the configuration page like this:
  Entity-ID: `http://saml.poc-consumer.localhost`
  Identity provider metadata: `<path to IDP metadata file>`
  Create new self-signed keystore? `Yes`
  Keystore password: `Test1234`
  Organization Name: `<Organization name>`
  Organization URL: `<Organization URL>`
  Technical email contact address: `<some email address>`
  Enable artifact consumer? `Yes`
  Enable Redirect consumer? `Yes`
  Enable SOAP Single Logout? `Yes`
  Enable OCES Attribute Profile? `Yes`
- Click the "`Configure system`" button
- Now the folder `~/.oiosaml-poc-consumer` has been created and populated with a number of files and directories.
- Configure the IdP with knowledge about the poc-consumer metadata. This is to be found in `~/.oiosaml-poc-consumer/metadata/SP/SPMetadata.xml`.

- For the SimpleSAMLphp IdP this requires converting the XML to PHP code (with a conversion service that is available in SimpleSAMLphp) and afterwards inserting this code into `metadata/saml20-sp-remote.php` file along with these extra attributes:

```
 'attributemap' => 'oiosaml',
  'NameIDFormat' => 'urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName',
  'SPNameQualifier' => '',
  'simplesaml.nameidattribute' => 'urn:oid:0.9.2342.19200300.100.1.1',
```

- Edit the file `~/.oiosaml-poc-consumer/oiosaml-sp.properties` and add these properties:

```
oiosaml-trust.bootstrap.base64=false
poc.provider=http://localhost:8080/poc-provider/ProviderService oiosaml-
trust.certificate.location=sts.jks oiosaml-
trust.certificate.password=Test1234
```

- Export a certificate from key private key of the STS (which was placed in the keystore `~/.oiosaml/TestVOCES1.pkcs12`) and import the certificate into a new keystore called `~/.oiosaml-poc-consumer/sts.jks`. The filename and password of the new keystore should match the value of the properties `oiosaml-trust.certificate.location` and `oiosaml-trust.certificate.password` in `~/.oiosaml-poc-consumer/oiosaml-sp.properties`.

```
keytool -exportcert -keystore ~/.oiosaml/TestVOCES1.pkcs12 -storetype pkcs12 -
alias "<alias>" -file sts.crt

keytool -importcert -keystore ~/.oiosaml-poc-consumer/sts.jks -file sts.crt -
alias sts -storepass Test1234
```

Testing the integration of the applications

Test 1 - basic flow

- Enter the URL http://localhost:8080/poc-consumer/
  ⇒ The start page of the WSC shows
- Click the link "Page requiring login"
  ⇒ The browser redirects to the login of the IdP (unless a login has already been performed)
- Perform login
  (In the case of using SimpleSAMLphp with the opensign module as authsource, the OpenSign applet shows and the user can browse for a private key file, e.g. TestMOCES1.pkcs12 and login using that) ⇒ Upon successful login, the browser gets a SAML User Assertion in a cookie and is redirected back to the poc-consumer which shows the assertion and a few more details
- Click the link in the bottom of the page "Call Service Provider with token"
  ⇒ A new page shows the EndPointReference and the Token (a new SAML assertion) from a request to the STS
- Click the link in the bottom of the page "Perform token WS request"
  ⇒ The consumer shows a page with data from the WS request to and response from the WSP. By default, the response is "null" but you may add "?length=3" to the URL of the page, and a real response object is generated (although the only thing visible is the default output of the toString() method). Note that a larger length (e.g. 6) generates an exponentially larger result object, so larger number can easily result in a very long response time, and possibly an OutOfMemoryError. (The length request parameter defines the depth of a dummy object graph).

Test 2 - request to interact

- Enter the URL http://localhost:8080/poc-consumer/
  ⇒ The start page of the WSC shows
- Click the link "Trigger Request to Interact"
  ⇒ The browser redirects to the login of the IdP (unless a login has already been performed)

- Perform login (as in Test 1)
  ⇒ Upon successful login, a page is shown that explains that extra information from the user is required to complete the request, and a more extensive link is provided
- Click the long link (which includes a ReturnToURL parameter)
  ⇒ The browser is redirected to a page of the WSP prompting the user for a 'secret'
- Enter some information in the field (anything, e.g. “1234”) and click the “Send” button
  ⇒The browser is now redirected back to a page of the WSC which shows the information that the user provided to the WSP.

In a real scenario, the WSP would probably not give away the secret info from the user to the WSC, but this is a simple way of simulating that the user approved the operation; ie. password or one-time-key has been entered by the user to the WSP, and thereby authorized the WSC to complete a certain webservice call at the WSP.

2.2 Configuring the .NET components

Please note, that there is no .NET STS included in the package. The Java STS could be used instead.

2.2.1 DLL Setup

Due to missing support for `sp:ProtectTokens` in WCF, Microsoft has released a few hotfixes for this particular issue. `System.ServiceModel` and `System.IdentityModel` have been patched to include this feature. Please note, though that these fixes might be considered non-standard with regards to MS-support. The hotfix includes support for `UseStrTransform`, which has been included in later WCF releases though, so please note, that you might have a potential DLL version conflict in the future on this particular issue!

The hotfixes cannot be downloaded from Microsoft, but are available on the OIOIDWS group on digitaliser.dk: http://digitaliser.dk/group/705156

2.2.2 Webservice provider

In this example, the webservice provider is hosted by an IIS on Windows Vista/Windows 7, but it should be possible to host in WAS too. It requires that both .NET 3.5 framework and Windows Identity Foundation 1.0 (WIF) are installed.

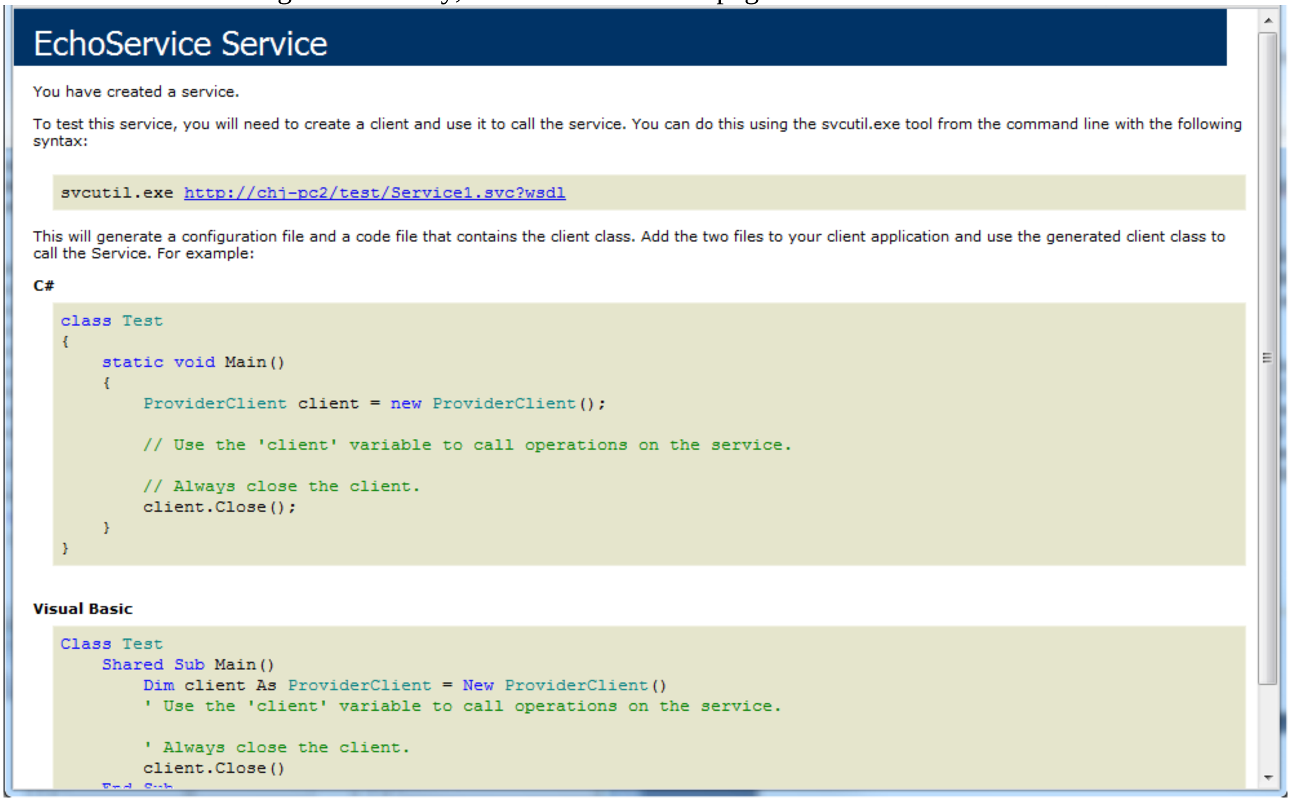
The steps described in this section explains what needs to be done on an IIS 7 running on a Windows 7. First, the Echo webservice itself should be configured:

- Open the IIS Manager
- Under Sites, “Default Web Site”, right click and select “Add Application”
- In the Alias field type “test”
- In the physical field, select the path to the EchoServiceProvider folder.
- Leave the application pool as Default

Now the user executing the webservice should install the STS certificate and grant the Echo service access to the STS certificate. This is done inside the MMC (Microsoft Management Console), File, Add/Remove Snap-in. In the list of available Snap-ins, select “Certificates”, click “Add >”, select “Computer account” and then finally click the Ok button.

Under the Console Root folder, a folder named "Certificates (Local Computer)" should be visible. Here, open the folders Personal, right click on the Certificates folder, select "All tasks" and "Import...". Point to the STS.pfx file, "click Next >", use Test1234 as password and enable "Mark this key as exportable". Then click "Next >", "Next >" and "Finish". Then the certificate is displayed in the list of personal certificates. Right click on the certificate, and select "All task > Manage Private Keys". Inside the permissions dialog, click "Add", type IIS_USRS and click "Ok" twice, and the service is able to access the certificate.

Check that the webservice is alive by entering http://localhost/test/Service1.svc in a browser. If the service and certificate are configured correctly, the browser shows a page like this:


EchoService Service

You have created a service.

To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the following syntax:

```
svcutil.exe http://chj-pc2/test/Service1.svc?wsdl
```

This will generate a configuration file and a code file that contains the client class. Add the two files to your client application and use the generated client class to call the Service. For example:

C#

```
class Test
{
    static void Main()
    {
        ProviderClient client = new ProviderClient();

        // Use the 'client' variable to call operations on the service.

        // Always close the client.
        client.Close();
    }
}
```

Visual Basic

```
Class Test
    Shared Sub Main()
        Dim client As ProviderClient = New ProviderClient()
        ' Use the 'client' variable to call operations on the service.

        ' Always close the client.
        client.Close()
```

2.2.3 Rich client

The rich client requires .NET 3.5 framework and Windows Identity Foundation 1.0 (WIF). The STS certificate should be installed in the Local Machine certificate store as described in the web service provider section above, in order to be able to select it in the dropdown-box. A few settings in WPFClient.exe.config can be configured:

- Configures the default values for the 3 textboxes in the application:

```
<applicationSettings>
     <WPFClient.Properties.Settings>
          <setting name="LocalSTSUrl" serializeAs="String">
               <value>https://tri-
test6.tritest6domain.com/adfs/services/trust/2005/usernamebasictransport</value>
          </setting>
          <setting name="STSUrl" serializeAs="String">
               <value>http://localhost:8082/sts/TokenService</value>
          </setting>
          <setting name="WSUrl" serializeAs="String">
               <value>http://localhost/test/Service1.svc</value>
```

```
        </setting>
    </WPFClient.Properties.Settings>
   </applicationSettings>
```

- Configuration of WIF logging, using normal .NET trace listener configuration:

```
<source name="System.ServiceModel"
        switchValue="Verbose, ActivityTracing"

        propagateActivity="true">
    <listeners>
      <add name="sdt"
         type="System.Diagnostics.XmlWriterTraceListener"
         initializeData= "c:\temp\wiftrace.e2e" />
    </listeners>
   </source>
```

2.2.4 IDP using AD FS 2.0

Configuring an ADFS is beyond the scope of this document, just a few overall notes on how to use it in our scenario though.

Instead of creating our own bootstrap token, it is possible to use AD FS to issue these tokens. The AD FS in that case takes the role as an IDP that authenticates users by means of username/password, Windows Authentication token, X509-certificate etc.

In case you want to test the rich .net client with an ADFS 2.0 server, it requires a Windows Server 2008 R2 with ADFS 2.0 Server installed as add on to an Active Directory installation. There are several ways in which you can authenticate against the ADFS, in the sample code two examples of authentication are implemented: Windows authentication and username/password. Depending on which method should be used, you should modify the `Window1.xaml.cs` file and thereby use the appropriate `GetTokenFromSTSUsingXXX` method (in the buttonWS_Click() method).

In the example code, an ADFS server is installed on the URL http://tri-test6.tritest6domain.com. Depending on the binding used for obtaining the bootstrap token from the ADFS, the url should be modified according to the binding.

On the ADFS, a signing certificate must be installed, and the public key of this signing certificate should be available for the rich client in the Windows certificate store. In the sample source code, the signing certificate is the one with the subject `CN=ADFS Signing - tri-test6.tritest6domain.com.`